



Pflanzenwächter für die Fensterbank Teil 3 – Erweiterung der Bodenfeuchtesensoren auf maximal sechs Stück.

In dem dritten Teil der Reihe „Pflanzenwächter“ kommen wir auf ein, in den vorherigen Teilen bereits implementiertes schaltungstechnisches Detail zurück und erweitern, die Anzahl der an unseren Pflanzenwächter anschließbaren Feuchtesensoren auf maximal sechs. Dem einen oder anderen sind vielleicht die bereits vorhandenen sechs Spannungsteiler aufgefallen, wobei bisher jedoch nur einer davon beschaltet war. Die restlichen 5 Spannungsteiler dienten bisher in der Hauptsache dazu, 5 von 6 Analog-Digital Eingänge des ESP32 definiert auf 0 zu ziehen um sie einerseits an einem evtl. unkontrolliertem Schwingen zu hindern, aber auch damit für unseren ESP feststellbar zu machen, an welchen der 6 Eingänge nun ein Feuchtesensor angeschlossen ist und an welchem nicht. Denn es soll, OHNE Änderung des Codes durch einfaches Anschließen eines weiteren Sensors (bis maximal 6) und einem anschließenden Reset des Systems möglich sein, weitere Sensoren nach Belieben und Notwendigkeit anzubinden. Wir können also eine beliebige Anzahl von mindestens 1 bis maximal 6 Sensoren, beginnend mit dem linken Port (GPIO36) folgend bis maximal zu dem rechten Port (GPIO 33) bestücken. Dabei dürfen jedoch keine Ports zwischen Sensoren ausgelassen werden, da diese andernfalls nicht erkannt werden. Nachfolgend ist die Maximalbestückung mit Feuchtesensoren gezeigt:



Unsere Teileliste erweitert sich somit auf:

- 1 x LED Farbe Grün (560nm); 5 mm
- 1x LED Farbe Gelb (605nm); 5 mm
- 1x LED Farbe Rot (633nm); 5 mm
- 6x130kΩ Widerstand Toleranz ±1%;
- 6x 47kΩ Widerstand Toleranz ±1
- 3x 150Ω Widerstand Toleranz ±1%;
- 6x [Kapazitiver Feuchtesensor](#)
- 1x ESP32-38Pin Variante Generic; Typ NodeMCU-32S; Beinchen 38;
- 1x YwRobot Breadboard Spannungsversorgung

Nachdem wir nun die nötigen Hardwareänderungen durchgeführt haben, müssen wir noch unseren Code auf dem ESP ein wenig anpassen sowie unsere Handy-APP erweitern. Zuerst laden wir folgenden Code auf unseren ESP hoch:

```
#include <driver/adc.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

// Portedefinieren Led's
#define LED_Rot    18 // Rote LED
#define LED_Gelb   14 // Gelbe LED
#define LED_Gruen  15 // Gruene LED

// LED PWM Einstellungen
#define PWMFreq 5000 // 5 Khz basisfrequenz für LED Anzeige
#define PWMledChannelA 0
#define PWMledChannelB 1
#define PWMledChannelC 2
#define PWMresolution 8 // 8 Bit Resolution für LED PWM

//Sonstige Definitionen
#define ADCAttenuation ADC_ATTEN_DB_11 //ADC_ATTEN_DB_11 = 0-3,6V Dämpfung ADC (ADC Erweiterung
#define MoistureSens_Poll_Interval 300000 // Intervall zwischen zwei Bodenfeuchtemessungen in Millisekunden -> alle 5
Minuten Datenpaket an Handy senden
#define MaxSensors 6 // Maximale Anzahl an anschließbaren FeuchteSensoren
#define StartInit true
#define Sens_Calib true
#define Sens_NOTCalib false

// Blynk APP Definitionen
#define BLYNK_GREEN "#23C48E"
#define BLYNK_BLUE "#04C0F8"
#define BLYNK_YELLOW "#ED9D00"
#define BLYNK_RED "#D3435C"
#define BLYNK_BLACK "#000000"
#define BLYNK_PRINT Serial
#define BLYNK_NO_BUILTIN
#define BLYNK_NO_FLOAT
// #define BLYNK_DEBUG

struct MoistureSensorCalibrationData
{
    int Data[MaxSensors*2] = {0,0,0,0,0,0,0,0,0,0,0,0}; // Calibration Data für Feuchtigkeitssensor. Bitte Projekt Text beachten
und Werte entsprechend anpassen
    String SensorName[MaxSensors] = {"Pflanze 1","Pflanze 2","Pflanze 3","Pflanze 4","Pflanze 5","Pflanze 6"}; // Sensorname,
der auch in der APP als Überschrift angezeigt wird
    byte StatusBorderPercentValues[MaxSensors*2][2]= { {10,50}, // Zweidimensionales Array für Prozentgrenzwerte
(Ampel) jeweils Einzeln pro Feuchtesensor (1 -6)
                {10,50},
                {10,50},
```

```

        {10,50},
        {10,50},
        {10,50}};

};

struct MoistureSensorData
{
    int Percent[MaxSensors] = {0,0,0,0,0,0}; // Feuchtigkeitssensordaten in Prozent
    byte Old_Percent[MaxSensors] = {0,0,0,0,0,0}; // Vorherige _ Feuchtigkeitssensordaten in Prozent (Zweck: DatenMenge
    einsparen.)
    bool DataValid [MaxSensors] = {false,false,false,false,false,false};
};

//Global Variables

char auth[] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"; // Hier lt. Anleitung Auth Token dere Blynk App eintragen (E-Mail).
// Deine WiFi Zugangsdaten.
char ssid[] = "Deine_WLAN_SSID"; // Bitte an eigene WLAN SSID anpassen
char pass[] = "Dein_WLAN_Passwort!"; // Bitte an eigene WLAN Passwort anpassen

MoistureSensorCalibrationData MCalib;
MoistureSensorData MMeasure;
byte AttachedMoistureSensors; // Detected Active Moisture Sensors (Count)
unsigned long Moisire_ServiceCall_Handler = 0; // Delay Variable for Delay between Moisire Readings

void setup() {
    pinMode(LED_Rot,OUTPUT);
    pinMode(LED_Gelb,OUTPUT);
    pinMode(LED_Gruen,OUTPUT);
    Serial.begin(115200); // initialize serial communication at 115200 bits per second:
    ledcSetup(PWMledChannelA, PWMfreq, PWMresolution);
    ledcSetup(PWMledChannelB, PWMfreq, PWMresolution);
    ledcSetup(PWMledChannelC, PWMfreq, PWMresolution);
    ledcAttachPin(LED_Rot, PWMledChannelA); // attach the channel to the GPIO to be controlled
    ledcAttachPin(LED_Gelb, PWMledChannelB);
    ledcAttachPin(LED_Gruen, PWMledChannelC);
    SetLedConfig(255,255,255);
    Serial.println(F("Systemkonfiguration:"));
    AttachedMoistureSensors = DetectMoistureSensors();
    Serial.print(AttachedMoistureSensors);
    Serial.println(F(" Bodenfeuchtigkeitssensor(en)"));
    Serial.print(F("Verbindung zu WLAN"));
    delay(500);
    Blynk.begin(auth, ssid, pass); // Initalize WiFi-Connection over Blynk Library
    Serial.println(F(" erfolgreich."));
    SetLedConfig(0,0,0);
    Init_Blynk_APP();
    Run_MoistureSensors(StartInit);
    for (int i = AttachedMoistureSensors;i < 6;i++) { Update_Blynk_APP(i,Sens_Calib); }
}

byte DetectMoistureSensors ()
{
    #define MinSensorValue 100
    byte Detected = 0;
    for (int i = 0;i < MaxSensors;i++)
    {
        int MSensorRawValue = ReadMoistureSensor_Raw_Val(i);
        if ( MSensorRawValue > MinSensorValue) { Detected++; } else {break;}
    }
    if (Detected < 1)
    {
        Serial.println(F("Keine Bodenfeuchtigkeitssesoren erkannt. System angehalten."));
        esp_deep_sleep_start();
        while(1) {}
    }
    return Detected;
}

bool SetLedConfig(byte Red,byte yellow,byte green)
{
    ledcWrite(PWMledChannelA, Red); // Rote LED
    ledcWrite(PWMledChannelB, yellow); // Gelbe LED
    ledcWrite(PWMledChannelC, green); // Gruene LED
    return true;
}

```

```

int ReadMoistureSensor_Raw_Val(byte Sensor)
{
    int ReturnValue,i;
    long sum = 0;
    #define NUM_READS 6
    adc1_config_width(ADC_WIDTH_BIT_12); //Range 0-4095
    switch (Sensor)
    {
        case 0:
        {
            adc1_config_channel_atten(ADC1_CHANNEL_0,ADCAttenuation);
            for (i = 0; i < NUM_READS; i++){ // Averaging algorithm
                sum += adc1_get_raw( ADC1_CHANNEL_0 ); //Read analog
            }
            ReturnValue = sum / NUM_READS;
            break;
        }
        case 1:
        {
            adc1_config_channel_atten(ADC1_CHANNEL_3,ADCAttenuation);
            for (i = 0; i < NUM_READS; i++){ // Averaging algorithm
                sum += adc1_get_raw( ADC1_CHANNEL_3 ); //Read analog
            }
            ReturnValue = sum / NUM_READS;
            break;
        }
        case 2:
        {
            adc1_config_channel_atten(ADC1_CHANNEL_6,ADCAttenuation);
            for (i = 0; i < NUM_READS; i++){ // Averaging algorithm
                sum += adc1_get_raw( ADC1_CHANNEL_6 ); //Read analog
            }
            ReturnValue = sum / NUM_READS;
            break;
        }
        case 3:
        {
            adc1_config_channel_atten(ADC1_CHANNEL_7,ADCAttenuation);
            for (i = 0; i < NUM_READS; i++){ // Averaging algorithm
                sum += adc1_get_raw( ADC1_CHANNEL_7 ); //Read analog
            }
            ReturnValue = sum / NUM_READS;
            break;
        }
        case 4:
        {
            adc1_config_channel_atten(ADC1_CHANNEL_4,ADCAttenuation);
            for (i = 0; i < NUM_READS; i++){ // Averaging algorithm
                sum += adc1_get_raw( ADC1_CHANNEL_4 ); //Read analog
            }
            ReturnValue = sum / NUM_READS;
            break;
        }
        default:
        {
            adc1_config_channel_atten(ADC1_CHANNEL_5,ADCAttenuation);
            for (i = 0; i < NUM_READS; i++){ // Averaging algorithm
                sum += adc1_get_raw( ADC1_CHANNEL_5 ); //Read analog
            }
            ReturnValue = sum / NUM_READS;
            break;
        }
    }

    return ReturnValue;
}

void Init_Blynk_APP()
{
    Blynk.setProperty(V1,"label",MCalib.SensorName[0]);
    Blynk.setProperty(V2,"label",MCalib.SensorName[1]);
    Blynk.setProperty(V3,"label",MCalib.SensorName[2]);
    Blynk.setProperty(V4,"label",MCalib.SensorName[3]);
    Blynk.setProperty(V5,"label",MCalib.SensorName[4]);
    Blynk.setProperty(V6,"label",MCalib.SensorName[5]);
}

```

```

void Update_Local_Display()
{
  byte red1 = 0;
  byte yellow1 = 0;
  byte green1 = 0;
  for (byte i = 0; i < AttachedMoistureSensors; i++)
  {
    if (MMeasure.DataValid[i])
    {
      if ( MMeasure.Percent[i] > MCalib.StatusBorderPercentValues[i][1])
      {
        green1++;
      } else if ( MMeasure.Percent[i] > MCalib.StatusBorderPercentValues[i][0])
      {
        yellow1++;
      } else
      {
        red1++;
      }
    }
  }
  if (red1 > 0)
  { SetLedConfig(255,0,0); }
  else if (yellow1 > 0)
  { SetLedConfig(0,255,0); }
  else if (green1 > 0)
  { SetLedConfig(0,0,100); }
  else
  { SetLedConfig(100,100,100); }
}

void Update_Blynk_APP(byte Sensor,bool Calibrated)
{
  switch (Sensor)
  {
    case 0:
    {
      if ((MMeasure.DataValid[0]) & (Calibrated))
      {
        if ( MMeasure.Percent[0] > MCalib.StatusBorderPercentValues[0][1])
        {
          Blynk.setProperty(V1,"color",BLYNK_GREEN);
        } else if ( MMeasure.Percent[0] > MCalib.StatusBorderPercentValues[0][0])
        {
          Blynk.setProperty(V1,"color",BLYNK_YELLOW);
        } else
        {
          Blynk.setProperty(V1,"color",BLYNK_RED);
        }
      }
      delay(100);
      Blynk.virtualWrite(V1,MMeasure.Percent[0]);
    } else
    {
      if (Calibrated)
      {
        Blynk.setProperty(V1,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V1,0);
        delay(100);
        Blynk.setProperty(V1,"color",BLYNK_BLACK);
      }
      else
      {
        Blynk.virtualWrite(V1,0);
        delay(100);
        Blynk.setProperty(V1,"color",BLYNK_BLUE);
      }
    }
    break;
  }
  case 1:
  {
    if ((MMeasure.DataValid[1]) & (Calibrated))
    {
      if ( MMeasure.Percent[1] > MCalib.StatusBorderPercentValues[1][1])
      {

```

```

        Blynk.setProperty(V2,"color",BLYNK_GREEN);
    } else if ( MMeasure.Percent[1] > MCalib.StatusBorderPercentValues[1][0])
    {
        Blynk.setProperty(V2,"color",BLYNK_YELLOW);
    } else
    {
        Blynk.setProperty(V2,"color",BLYNK_RED);
    }
    delay(100);
    Blynk.virtualWrite(V2,MMeasure.Percent[1]);
} else
{
    if (Calibrated)
    {
        Blynk.setProperty(V2,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V2,0);
        delay(100);
        Blynk.setProperty(V2,"color",BLYNK_BLACK);
    }
    else
    {
        Blynk.virtualWrite(V2,0);
        delay(100);
        Blynk.setProperty(V3,"color",BLYNK_BLUE);
    }
}
break;
}
case 2:
{
    if ((MMeasure.DataValid[2]) & (Calibrated))
    {
        if ( MMeasure.Percent[2] > MCalib.StatusBorderPercentValues[2][1])
        {
            Blynk.setProperty(V3,"color",BLYNK_GREEN);
        } else if ( MMeasure.Percent[2] > MCalib.StatusBorderPercentValues[2][0])
        {
            Blynk.setProperty(V3,"color",BLYNK_YELLOW);
        } else
        {
            Blynk.setProperty(V3,"color",BLYNK_RED);
        }
        delay(100);
        Blynk.virtualWrite(V3,MMeasure.Percent[2]);
    } else
    {
        if (Calibrated)
        {
            Blynk.setProperty(V3,"label","Deaktiviert");
            delay(100);
            Blynk.virtualWrite(V3,0);
            delay(100);
            Blynk.setProperty(V3,"color",BLYNK_BLACK);
        }
        else
        {
            Blynk.virtualWrite(V3,0);
            delay(100);
            Blynk.setProperty(V3,"color",BLYNK_BLUE);
        }
    }
}
break;
}
case 3:
{
    if ((MMeasure.DataValid[3]) & (Calibrated))
    {
        if ( MMeasure.Percent[3] > MCalib.StatusBorderPercentValues[3][1])
        {
            Blynk.setProperty(V4,"color",BLYNK_GREEN);
        } else if ( MMeasure.Percent[3] > MCalib.StatusBorderPercentValues[3][0])
        {
            Blynk.setProperty(V4,"color",BLYNK_YELLOW);
        } else
        {
            Blynk.setProperty(V4,"color",BLYNK_RED);
        }
    }
}

```

```

    }
    delay(100);
    Blynk.virtualWrite(V4,MMeasure.Percent[3]);
  } else
  {
    if (Calibrated)
    {
      Blynk.setProperty(V4,"label","Deaktiviert");
      delay(100);
      Blynk.virtualWrite(V4,0);
      delay(100);
      Blynk.setProperty(V4,"color",BLYNK_BLACK);
    }
    else
    {
      Blynk.virtualWrite(V4,0);
      delay(100);
      Blynk.setProperty(V4,"color",BLYNK_BLUE);
    }
  }
  break;
}
case 4:
{
  if ((MMeasure.DataValid[4]) & (Calibrated))
  {
    if ( MMeasure.Percent[4] > MCalib.StatusBorderPercentValues[4][1])
    {
      Blynk.setProperty(V5,"color",BLYNK_GREEN);
    } else if ( MMeasure.Percent[4] > MCalib.StatusBorderPercentValues[4][0])
    {
      Blynk.setProperty(V5,"color",BLYNK_YELLOW);
    } else
    {
      Blynk.setProperty(V5,"color",BLYNK_RED);
    }
    delay(100);
    Blynk.virtualWrite(V5,MMeasure.Percent[4]);
  } else
  {
    if (Calibrated)
    {
      Blynk.setProperty(V5,"label","Deaktiviert");
      delay(100);
      Blynk.virtualWrite(V5,0);
      delay(100);
      Blynk.setProperty(V5,"color",BLYNK_BLACK);
    }
    else
    {
      Blynk.virtualWrite(V5,0);
      delay(100);
      Blynk.setProperty(V5,"color",BLYNK_BLUE);
    }
  }
  break;
}
default:
{
  if ((MMeasure.DataValid[5]) & (Calibrated))
  {
    if ( MMeasure.Percent[5] > MCalib.StatusBorderPercentValues[5][1])
    {
      Blynk.setProperty(V6,"color",BLYNK_GREEN);
    } else if ( MMeasure.Percent[5] > MCalib.StatusBorderPercentValues[5][0])
    {
      Blynk.setProperty(V6,"color",BLYNK_YELLOW);
    } else
    {
      Blynk.setProperty(V6,"color",BLYNK_RED);
    }
    delay(100);
    Blynk.virtualWrite(V6,MMeasure.Percent[5]);
  } else
  {
    if (Calibrated)
    {

```



```

        Blynk.setProperty(V6,"label","Deaktiviert");
        delay(100);
        Blynk.virtualWrite(V6,0);
        delay(100);
        Blynk.setProperty(V6,"color",BLYNK_BLACK);
    }
    else
    {
        Blynk.virtualWrite(V6,0);
        delay(100);
        Blynk.setProperty(V6,"color",BLYNK_BLUE);
    }
}
break;
} // End Switch
}

void Get_Moisture_DatainPercent()
{
    byte CalibDataOffset = 0;
    for (byte i = 0; i < AttachedMoistureSensors; i++)
    {
        CalibDataOffset = i * 2;
        int RawMoistureValue = ReadMoistureSensor_Raw_Val(i);
        if ((MCalib.Data[CalibDataOffset] == 0) || (MCalib.Data[CalibDataOffset+1] == 0)) // MinADC Value maxADC ADC Value
        {
            MMeasure.Percent[i] = RawMoistureValue;
            MMeasure.DataValid[i] = false;
        } else
        {
            RawMoistureValue = MCalib.Data[CalibDataOffset+1] - RawMoistureValue;
            RawMoistureValue = MCalib.Data[CalibDataOffset] + RawMoistureValue;
            MMeasure.Percent[i] = map(RawMoistureValue, MCalib.Data[CalibDataOffset], MCalib.Data[CalibDataOffset+1], 0, 100);
            if ((MMeasure.Percent[i] > 100) || (MMeasure.Percent[i] < 0))
            {
                MMeasure.Percent[i] = RawMoistureValue;
                MMeasure.DataValid[i] = false;
            } else { MMeasure.DataValid[i] = true; }
        }
    }
    return ;
}

void Run_MoistureSensors (bool Init) // Hauptfunktion zum Betrieb der Bodenfeuchtesensoren
{
    byte MinSensValue = 100;
    if ((millis() - Moisture_ServiceCall_Handler >= MoistureSens_Poll_Interval) || (Init))
    {
        Moisture_ServiceCall_Handler = millis();
        Get_Moisture_DatainPercent();
        for (int i = 0; i < AttachedMoistureSensors; i++)
        {
            if (MMeasure.DataValid[i])
            {
                if (MMeasure.Percent[i] != MMeasure.Old_Percent[i])
                {
                    MMeasure.Old_Percent[i] = MMeasure.Percent[i];
                    if (MMeasure.Percent[i] < MinSensValue) { MinSensValue = MMeasure.Percent[i]; };
                    Serial.print(F("Feuchtigkeitswert Sensor "));
                    Serial.print(i);
                    Serial.print(F(" in Prozent :"));
                    Serial.print(MMeasure.Percent[i]);
                    Serial.println(F(" %"));
                    Update_Blynk_APP(i, Sens_Calib); // Aktualisiere Handywerte
                }
            } else
            {
                Update_Blynk_APP(i, Sens_NOTCalib); // Aktualisiere Handywerte
                Serial.print(F("Sensor "));
                Serial.print(i);
                Serial.print(F(" nicht kalibriert. Bitte kalibrieren. Rohdatenwert:"));
                Serial.println(MMeasure.Percent[i]);
            }
        }
    }
}

```

```
    }  
  }  
  Update_Local_Display();    // Aktualisiere lokales Pflanzenwächter Display (Led)  
}  
  
// Main Loop  
void loop()  
{  
  Run_MoistureSensors(false);  
  Blynk.run(); // Execute Blynk Basic- Functions  
}
```

Es ist vor dem hochladen auf unseren ESP zu beachten, das folgende Parameter/Codezeilen an die jeweiligen Bedürfnisse angepasst werden **müssen**:

```
int Data[MaxSensors*2]={0,0,0,0,0,0,0,0,0,0};
```

[Werte bitte gemäß Beschreibung in Teil 1 der Reihe anpassen.](#)

```
#define MoistureSens_Poll_Interval 300000
```

Intervall zwischen zwei Bodenfeuchtemessungen in Millisekunden. **Der Wert hat darauf Einfluss, wie oft die Handydaten aktualisiert werden, und damit auch, welches Datenvolumen für die Übertragung pro Zeit anfällt.** Je höher, desto größeres Intervall. Bitte auf einen Wert setzen, der zu dem eigenen Datenvolumen bzw. Budget passt. [Beispiel: 300000 ms = 5 Minuten Datenübertragungsintervall]

```
char auth[]="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";  
char ssid[]="Deine_WLAN_SSID";  
char pass[]="Dein_WLAN_Passwort!";
```

[Werte bitte gemäß Beschreibung in Teil 2 der Reihe anpassen.](#)

Es **können** folgende Parameter/Codezeilen, an die jeweiligen Bedürfnisse angepasst werden:

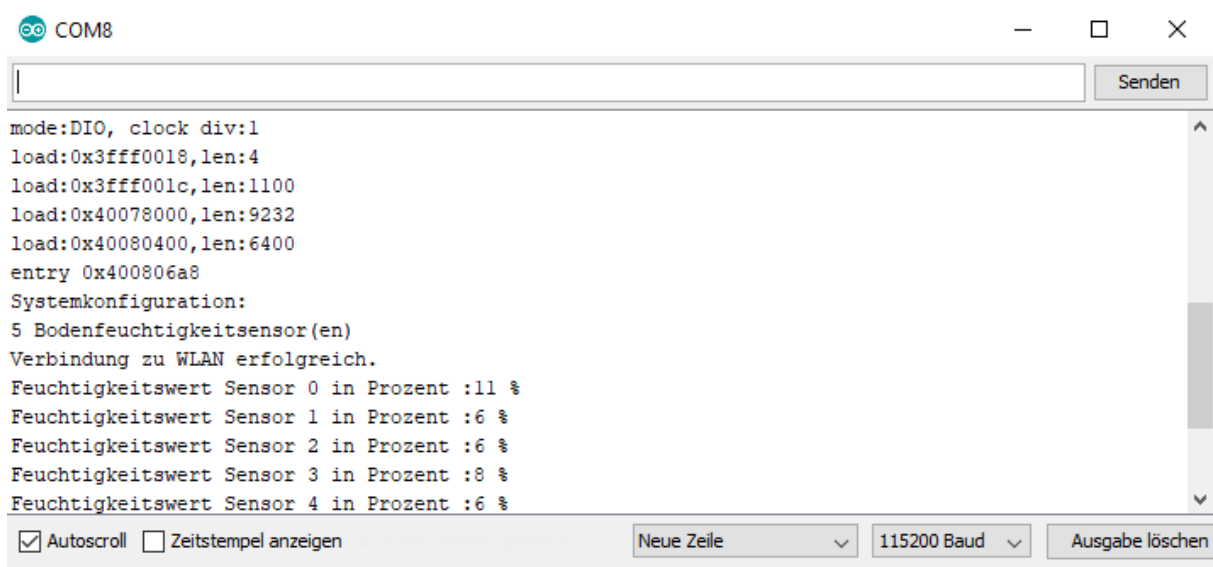
```
String SensorName[MaxSensors]={"Pflanze1","Pflanze2","Pflanze3","Pflanze4","Pflanze5","Pflanze6"};
```

Feuchtesensorenname, der in der APP als Überschrift angezeigt wird.

```
byte StatusBorderPercentValues[MaxSensors*2][2]={ {10,50}, ....
```

Zweidimensionales Array für Prozentgrenzwerte (Ampel) jeweils einzeln pro Feuchtesensor (1 -6). Hat Einfluss auf die „Ampel“ Anzeige und die Anzeige der Werte in der APP. Erster Wert (10) gibt die Übergangsgrenze zwischen Status „rot“ und Status „gelb“ an. Zweiter Wert gibt die Übergangsgrenze zwischen Status „gelb“ und Status „grün“ an. Beispiel: ab 51 % Bodenfeuchte „grün“ ab 9% Bodenfeuchte „rot“

Wenn alle genannten Parameter korrekt definiert worden sollte die Ausgabe auf dem seriellen Monitor nun wie folgt aussehen:

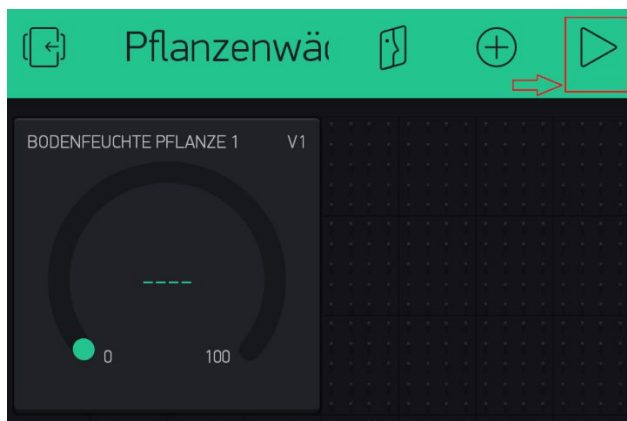


```
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:9232
load:0x40080400,len:6400
entry 0x400806a8
Systemkonfiguration:
5 Bodenfeuchtigkeitssensor(en)
Verbindung zu WLAN erfolgreich.
Feuchtigkeitsswert Sensor 0 in Prozent :11 %
Feuchtigkeitsswert Sensor 1 in Prozent :6 %
Feuchtigkeitsswert Sensor 2 in Prozent :6 %
Feuchtigkeitsswert Sensor 3 in Prozent :8 %
Feuchtigkeitsswert Sensor 4 in Prozent :6 %
```

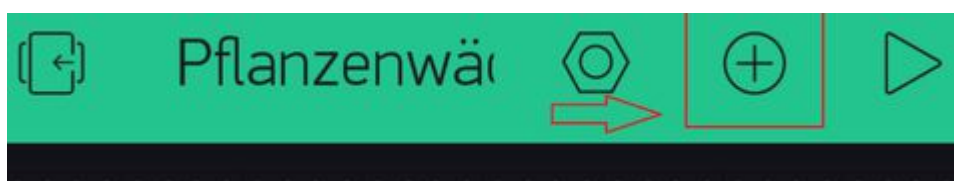
Hinweis: In diesem Beispiel sind statt max6. nur 5 Sensoren angeschlossen

Als nächstes müssen wir unsere HandyApp noch auf die veränderte Hardware und Firmware anpassen.

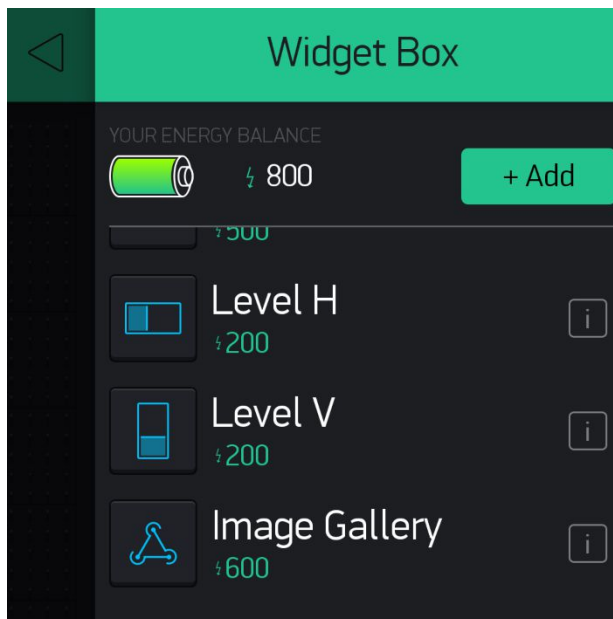
Da wir in unserer Blynk HandyAPP nur 2000 Energiepunkte kostenlos zur Verfügung gestellt bekommen, müssen wir bei 6 angebundenen Sensoren etwas haushalten. Im ersten Schritt löschen wir daher das im zweiten Teil angelegte „Gauge“ Element.



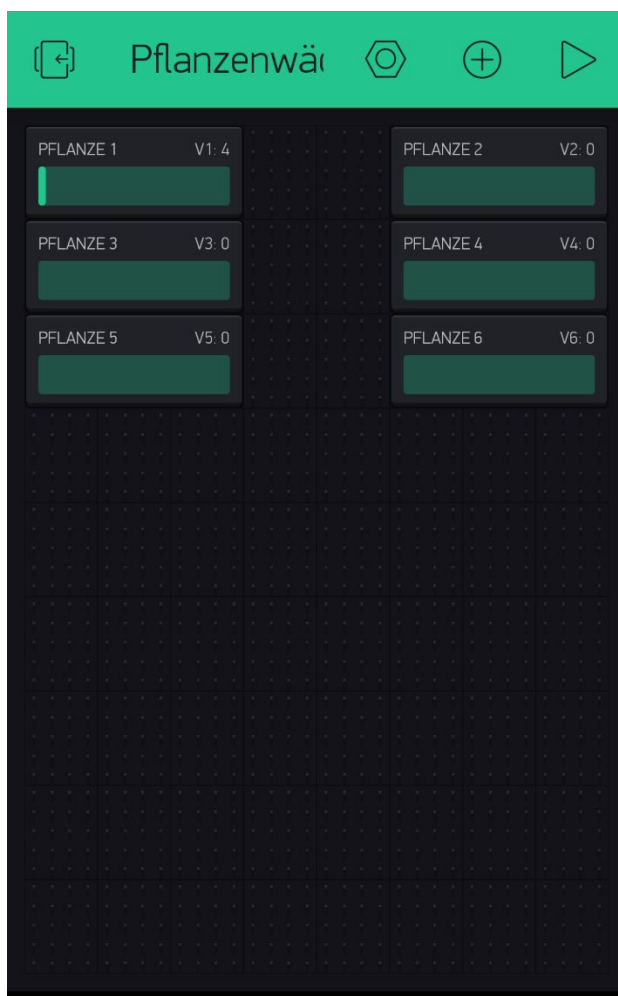
Das „Gauge“ Element ersetzen wir durch das „Level H“ Element



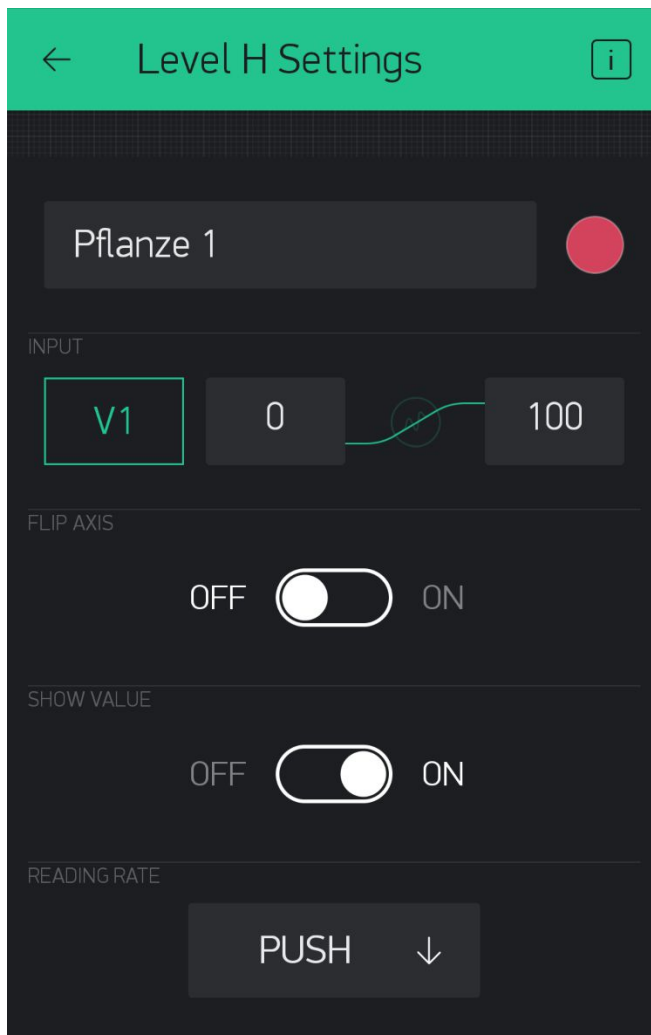
Nun fügen wir also zur Anzeige der aktuellen Bodenfeuchte das Energiegünstigere Element „Level H“ insgesamt 6-mal hinzu.



Nun sollte unsere Oberfläche ungefähr so aussehen:

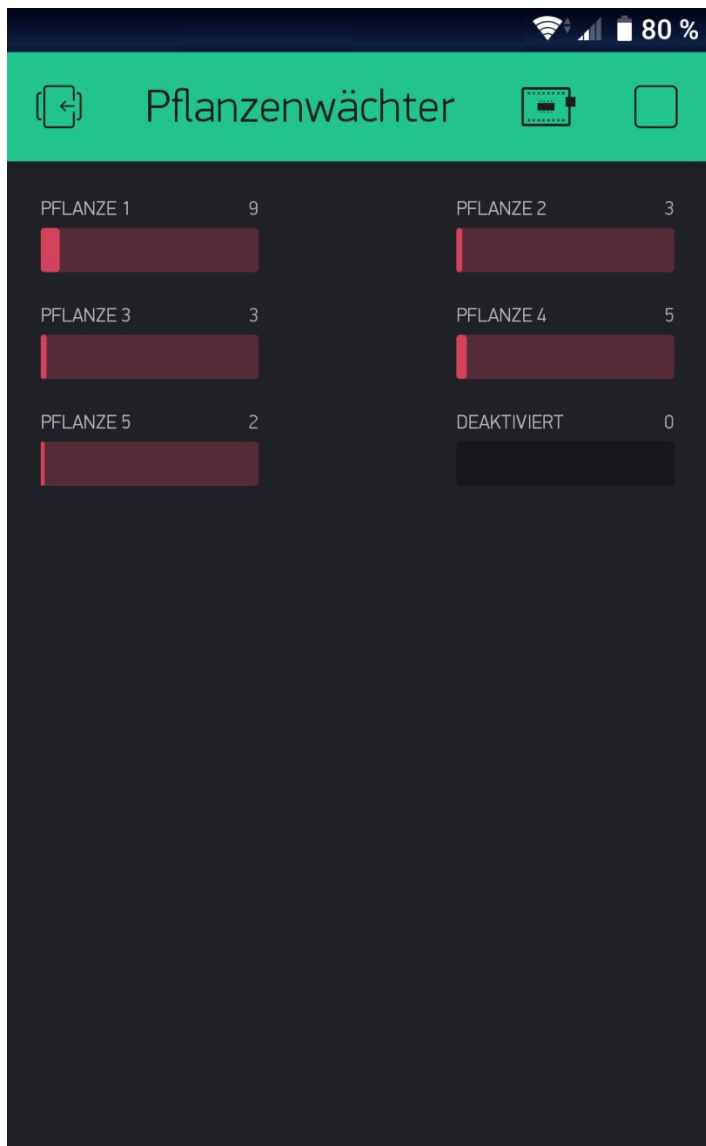


Nun müssen wir noch drei Werte jeweilig in den Eigenschaften der 6 Elemente anpassen. Exemplarisch zeige ich hier die Einstellungen für den ersten Sensor



Die wichtigste Anpassung ist hierbei die Input Variable. Diese muss fortlaufend für jedes „Level H“ Element angefangen von Input „V1“ für Pflanze 1 bis zu „V6“ für Pflanze 6 definiert werden. „V“ steht dabei für „Virtuelle Pins“ und ermöglichen eine Zuordnung zwischen den Elementen der APP und der Anweisungen im Code. Weiterhin wählen wir als Minimalwert 0 und als Maximalwert 100 aus. Als drittes definieren wir als Reading Rate „Push“ ein. Bei dem Design können eine beliebige Farbe, eine beliebige Schriftgröße und ein beliebiger Name gewählt werden, da diese Parameter später durch unser Programm auf dem ESP gesetzt bzw. definiert werden.

Das Endergebnis, bei Aktivierung des Projekts sollte nun ungefähr wie folgt aussehen:



(der letzte Sensor wird als deaktiviert dargestellt, da ich in dem Beispiel nur 5 Sensoren angeschlossen habe. Bei Verwendung von 6 Sensoren wird das Feld automatisch aktiviert)

Weitere Informationen über die Blynk APP und deren Verwendung in Controllern findest du unter

- Blynk Einführung -> <https://www.blynk.cc/getting-started>
- Dokumentation -> <http://docs.blynk.cc/>
- Sketch Generator -> <https://examples.blynk.cc/>
- Aktuellste Blynk Bibliothek -> https://github.com/blynk/blynk-library/releases/download/v0.6.1/Blynk_Release_v0.6.1.zip
- Aktuellster Blynk server -> <https://github.com/blynk/blynk-server/releases/download/v0.41.5/server-0.41.5.jar>
- Blynk Startseite -> <https://www.blynk.cc>

Ich wünsche viel Spaß beim Nachbauen, und bis zum nächsten Mal.

